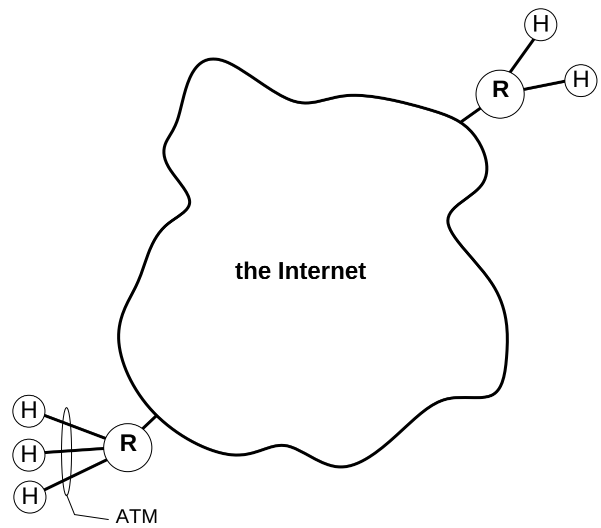
Learning from the Present — Things that IP got right and ATM got wrong

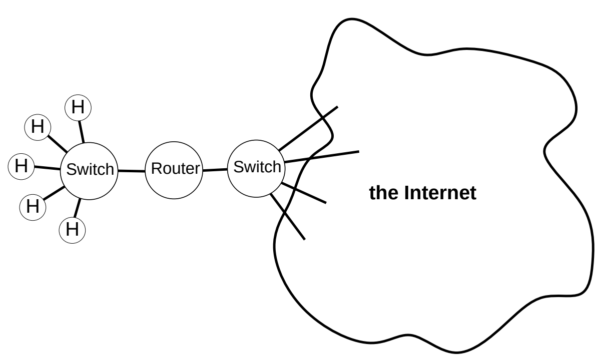
Van Jacobson
Lawrence Berkeley Laboratory
Berkeley, CA 94720

USENIX High-speed Networking Symposium
Berkeley, CA
1–3 August 1994

Note: I am *not* going to talk about ATM as an interconnection technology.



In particular, this is a perfectly reasonable picture:



Things that ATM got right

- ATM is better (cheaper, more flexible) than TDM for trunking.
- The 'bandwidth independence' of ATM is useful for host/network interfaces.

Hierarchy of networking problems:

- Going fast
- Getting big
- Crossing borders

(Difficulty increases going down. First item is hard; last is within ϵ of impossible.)

Designers should be careful that solutions at one level don't make problems at next level harder.

Getting big – traffic scaling

ATM 'call' (virtual circuit) model is a poor match to everything we know about data traffic.

VCs work when the call lifetime is long compared to the call setup time.

All Internet wide-area traffic studies have found that average long-haul connection transfers average 2–5KB (Paxson93, Danzig92, Klaffy93).

For a 1 Gbit transcontinental (120ms RTT) pipe, this means call lifetime should be 25μsec. but gets inflated by factor of 5000 by setup time.

This generates completely useless state for 4000 connections/trunk and requires at least one call completion every 25μsec. (40K/sec.).

Getting big – multicast

There's one case where current Internet traffic is long-lived compared to RTT: MBone voice & video.

Unfortunately, Internet voice & video success deeply tied to IP multicast model. ATM doesn't (and can't) support this model.

Getting big – multicast (cont.)

IP multicast model:

- Receivers announce interest.
- Senders just send.
- Network takes care of delivering data from senders to all interested receivers.

If everyone both sends & receives, this scales $O(logR)$. It works because multicast address has global meaning and provides network-level identity for session.

ATM multicast model:

- Sender knows every receiver, creates a call to one then ‘adds’ others to call.

If everyone sends & receives, this scales $O(R^2)$.

Getting big – reliability

ATM VC state is spread out over all switches in the path (VPI/VCI in cell is per-hop).

If hop’s reliability is λ, failure probability for n hop path is $1 - \lambda^n$. E.g., for typical router reliabilities of 10^{-4} (99.99% uptime), path failure probability for typical 22 hop Internet path is 1%.

For moderately well connected IP topology, path failure probability goes exponentially to zero with number of hops, independent of per-hop reliability.

Short summary:

- ATM fails if anything fails.
- IP fails if everything fails.

Getting big – summary

Central problem is that an ATM core element, the ‘call’, is *not* a low-level building block — it’s a very high-level abstraction. It is the wrong abstraction for a lot of problems and has poor scaling properties.

IP was built on a slight idealization of packet forwarding behavior that is intrinsic part of routing. It is *very* low level. I.e., if you can’t build a solution from this building block, you can’t build a solution.

Crossing boundaries – making promises

Part of organization’s willingness to carry transit traffic (erase boundary) is based on what kind of obligation they’re committing to.

IP router’s promise:

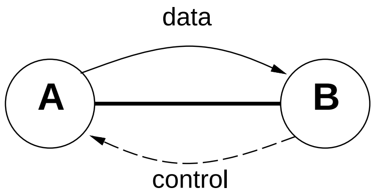
- I’ll try to send packet towards destination.

ATM switch’s promise:

- I’ll send cell out port that was in direction of destination at time call was set up.
- I won’t crash.
- I’ll remember your call until you tell me to forget.

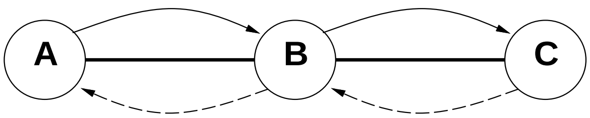
Crossing boundaries – traffic control

One reason for having boundary is desire to control what & how much crosses it. Usually this involves some sort of feedback loop:

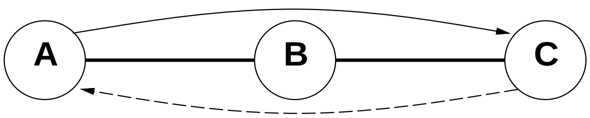


Crossing boundaries – traffic control (cont.)

There's a tendency in VC protocols to (incorrectly) generalize the simple case into hop-by-hop flow control:



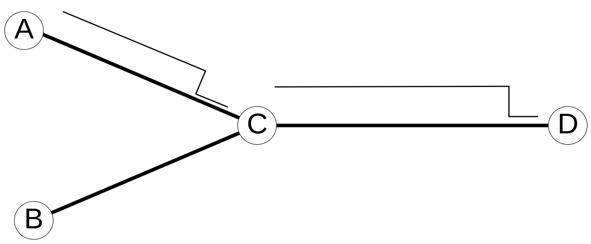
As Routh pointed out more than a century ago, the correct (and far more stable) generalization is end-to-end flow control:



Crossing boundaries – traffic control (cont.)

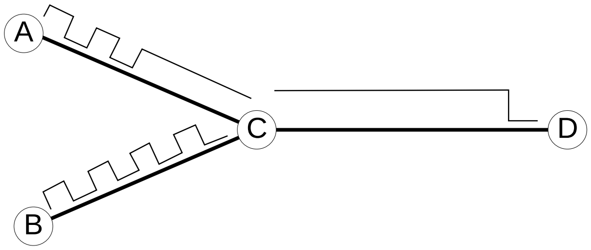
One (of many) problems with hop-by-hop flow control is that it converts a traffic problem anywhere into a traffic problem everywhere.

Say there's a source A sending to dest D at the capacity of links A–C and C–D:



Crossing boundaries – traffic control (cont.)

If a periodic source from B to D starts up, the queue at C must increase until the flow control is activated (since there's no excess capacity on the C–D link to dissipate the queue). The flow control will eventually gate A at the same frequency as the B traffic:



Cross traffic also inherits this pattern and, even if B shifts from periodic to steady, the pattern will persist.

Crossing boundaries – summary

There is a core philosophical difference between ATM and IP:

1. ATM: *Everything's* a boundary (UNI/NNI separation, service & provider ids in Q.931, etc.). By careful engineering and complex negotiation, it may be possible to send data across a boundary.

2. IP: *Nothing's* a boundary. By careful engineering and complex negotiation, it may be possible to not send data across a boundary.

If the object of this exercise is to communicate, (2) works much better than (1).